所属类别	2021 年“华数杯”全国大学生数学建模竞赛	参赛编号

进出口公司的货物装运策略研究

摘 要

随着经济社会的不断发展，物流行业也随之飞速发展，巨大的物流订单席卷而来，于是货运装配问题就显得日益重要．而传统的人工规划经验型货运配送逐渐凸显出其难以达到最优化配送和成本难以控制的缺陷，于是这就要求我们思考如何进一步地考虑如何设计数学模型对货机装配这一过程进行最优化方案设计．本文设计多种装配规则，同时设计了如何最大化空间分割以提高货舱空间利用率的方案，且进一步地考虑了市场的风险性与不确定性从而对货运装配的供货量进行有效把控，取得了良好效果．

针对问题一的经典三维装箱问题 (3D-KLP)，我们以货舱的体积利用率最大作为目标函数，设置重量约束、体积约束、三维尺寸摆放约束、重量平衡约束等多个约束条件．其次，对于货物的摆放规则，我们首先提出了占角策略的定位规则和密度升序的定序规则，其次设计了在空间编码的前提下针对剩余空间的三空间分割原则，然后利用 python 进行编程进行问题求解．最终我们得到了三架不同类型的货机的最优货运方案并进行了可视化，在保证体积利用率最大的前提下合理最大化地利用有限的货舱空间．

针对问题二，我们以集装箱与货舱的空间利用率为目标函数．首先对小型货物的集装箱装载进行设计，通过对集装箱的初筛，我们选取柔性航空集装箱，随后我们建立二维装箱模型，对同构同向的货物进行摆放，求解不同集装箱的空间利用率，最终确定将四种小型货物 HW2,HW6,HW7,HW10 均采用 Y7 柔性集装箱进行装配，并求解出集装箱货物的摆放方式，同时对货物数据进行更新．其次，我们改进第一问所建模型，通过增加约束条件条件，比较不同的定序规则，再次利用 python 编程求解，最终确定选取小型货机作为所选机型，且按照最长边递减的定序规则进行摆放，所需架次为 370 架，货机的空间利用率约为 13.9%．

针对问题三，为考虑经济效益，我们不再以空间利用率为目标函数，而是对货运的利润进行分析，我们引入了三种机型的附加成本 A，B，C，选取第二问中所需架次最少的货机配送方案，建立模型以控制货机运送成本最小，以得到实际的利润最大值，并确定出最终的货运运输方案．

针对问题四，我们首先绘制出每种货物历史需求量的 Q-Q 图，判断出其近似符合正态分布．通过观察所给 95%和 70%与正态分布 3_σ 原则中概率的相似性，给出可靠性的定义为货物历史需求量的可靠性，即实际运输时的每种货物需求量，只能取自 $(\mu-2\sigma,\mu+2\sigma)$ 的历史数据．我们再将实际需求量和货运总量离散化，假定为 $\mu-2\sigma,\mu,\mu+2\sigma$ 三种数值，并根据 3_σ 原则，给出实际需求量为这三种数值的概率．最后建立模型通过计算经济期望，比较得出最优经济期望下的装运方案、机型选择以及所需架次．

针对问题五，我们沿用了第四问的思路和方法进行求解，并得出最优经济期望下的装运方案、机型选择以及所需架次．

关键词：三维装箱问题; 最优化货运装配; 三空间分割规则; 定序规则；经济效益

目录

一、问题的提出

1.1 问题背景

随着物流行业的飞速发展，伴随着大批量物流订单的到来，货运配送问题引起了许多人的关注与思考．随着生产的多样化与不确定性和行业竞争激烈加剧，这对货物在飞机上的装配也提出了更高的要求，如何使空间利用率和载重利用率最大化，同时保障飞机的平稳飞行，同时还需考虑如何设计使得经济效益最大化，这都是目前研究的热点问题．

而目前货物的实际装载大多依靠人工的技能和经验，对于大规模的货物装载问题处理还亟待提升，研究三维装箱问题，对提高运输效率，降低运输成本，提升企业盈利能力等都有着十分积极的现实意义．而一般来说，包装问题是十分复杂的，有许多约束条件限制，重量、体积、材料等都加剧了问题的难度．

货物装箱方案的设计可以归纳为装箱问题．但装箱问题不仅可以用于车厢装载、物流运输方面的问题, 同样也适用于求解加工业中的板材切割、成品包装, 印刷业中的排版设计, 房屋建筑中的设施布局规划, 甚至管理业中的生产调度、资源分配等问题．研究装箱问题, 不仅有利于物流业的发展, 也对其他行业的发展有着重要影响, 具有突出的实际应用意义．

同时, 三维装箱问题属于组合优化的问题中 NP-hard 问题, 需要针对装箱过程中的边界约束、重量约束、体积约束和稳定性约束等众多种约束进行优化求解, 因此研究三维装箱也具有重要的理论价值．

1.2 问题重述

(1) 问题一要求我们按照前 50 个周期各种货物销售量的平均值来组织货源，然后利用大、中、小三种类型的货运飞机进行货运配送，设计出合适方案使得货运飞机尽量不留空隙．

(2) 问题二要求我们继续按照第一问同样的货源进行组织货源，同时将体积小于 $2m^3$ 的货物用集装箱装载，建立模型使得集装箱与货运飞机都尽量不留空隙，确定使用的机型，所需要的架次以及具体的装运方案．

(3) 问题三在问题二的基础上，调整以集装箱、飞机尽量不留空隙为基础的思路，进一步地考虑经济效益，从而尝试提出新的货运方案，满足公司的经济期望，并计算最佳利润．

(4) 问题四要求我们进一步考虑市场的不确定性与风险性，尝试在可靠性未 95% 下得到最大利润值，同时给出相应的货运装运策略．

(5) 问题五将问题四中的可靠性从 95% 变为 70%，要求我们重新计算最大利润值，并给出新的装运策略．

二、问题分析

2.1 问题一的分析

问题一是经典三维装箱问题（3D-KLP），我们以飞机货舱尽量不留空隙为目的，建立数学模型求解．我们选择货舱的体积利用率作为目标函数，求解目标函数的最大值，并得出相

应的摆放情况．为建立模型，我们首先对货舱进行建系，同时对空间进行编码处理．由于单个货物的重量较大，货舱运载量的主要限制条件为载重限制，故我们采用“密度递增”的定序规则和“占角策略”的定位规则，将密度最小的货物第一个放入原点所在的角落，之后再利用三空间分割规则对剩余空间划分，重复此过程，在货物摆放过程中，我们又设定了重量约束，货舱重量平衡约束，体积约束等约束条件，直到剩余空间不再支持继续放入货物，从而得出空间利用率以及货物摆放情况．

2.2 问题二的分析

第二问是三维装箱问题，我们仍是选择将集装箱和货舱空间利用率作为目标函数，求解目标函数的最大值，并得出相应的摆放情况，以及所需的飞机架次．集装箱非一次性用品，可循环使用，故不考虑其数量限制．我们首先通过比较选择柔性航空集装箱，将小于 $2m^3$ 的进行装箱，为运输方便，每种货物仅装填一种集装箱．我建立模型求解同构同向二维装箱问题，选择合适的集装箱，并计算新集装箱的体积、数量、重量，以代替原货物的指标．我们进而将问题一的模型进行改进，仍是考虑“密度递增”的定序规则，将集装箱和体积大于 $2m^3$ 的货物一并装入飞机，货物装尽时，计算空间利用率和飞机的架次．

2.3 问题三的分析

在第三问中，要求调整之前的货运策略，使得经济效益提高，以达到经济期望．于是在第一、二问的基础上，我们通过不同的定序规则把货物或集装箱进行摆放，例如最长边递减规则、密度规则、体积规则等等，来得到不同的货运方案．由于货物的数量不变，那么我们假设货物能够全部运走且货物的经济效益能够发挥到最大，即货物的利润为一个定值，则更多的经济效益要依赖于飞机所带来的的成本大小．由于飞机在空中的飞行成本由货物的运输成本决定，与飞机的选择无关．因此我们假定大、中、小三种类型的飞机由于装卸货、起降带来的附加成本为 A、B、C(A>B>C)，依靠于未定的 A、B、C 值和更少的飞机架次来确定最少成本．进而利用利润＝销售额-货物成本-运输成本-附加成本算出最大利润．

2.4 问题四的分析

针对问题四，我们首先对附件 2 中的每个货物的历史需求量进行分析，我们分析每种货物历史需求量的正态分布性，画出每种货物历史需求量的 Q-Q 图，发现每种货物的历史需求量均符合正态分布．同时，我们观察到问题四与问题五中可靠性 95%、70% 与正态分布中 3σ 原则中的概率 95.45% 和 68.27% 具有较大的相似性，因此我们定义可靠性 95% 的含义是指附件二中各种货物历史数据仅有 95％的数据可代表实际的运输情形，即实际的运输时的每种货物需求量，只能取自 $(\mu_i - 2\sigma_i, \mu_i + 2\sigma_i)$ 的历史数据，其中 μ 为历史数据的均值，σ 为历史数据的标准差，i 为不同的货物种类．我们将每种货物的实际需求量离散化，假定为三种数值，分别为 $\mu_i - 2\sigma_i$，μ_i，$\mu_i + 2\sigma_i$，分别设定其概率分布，同时，我们将所有货物的装运总量也只分为三种情况：每种货物的货运量分别为 $\mu_i - 2\sigma_i$，μ_i，$\mu_i + 2\sigma_i$ 件，建立模型计算经济期望．再通过比较，得出在最优经济期望下的装货方式、机型的选择以及所需架次．

2.5 问题五的分析

我们仍是按照问题四的思路，首先确定可靠性 70%的含义，即货物历史数据仅有 70%的数据可代表实际的运输情形，即实际运输时的每种货物需求量，只能取自 $\mu_i-\sigma_i,\mu_i+\sigma_i$ 的历史数据．我们仍是将每种货物的实际需求量离散化，假定为三种数值，分别为 $\mu_i-\sigma_i$，μ_i，$\mu_i+\sigma_i(i$ 为货物种类)，设定其概率分布．同时，我们将所有货物的装运总量也只分为三种情况：每种货物均只装 $\mu_i-\sigma_i$，μ_i，$\mu_i+\sigma_i$ 件，建立模型计算经济期望．再通过比较，得出在最优经济期望下的装货方案、机型的选择以及所需架次．

三、模型假设

1. 机舱内与集装箱内货物摆放位置无区域限制
2. 货物均为长方体型且质量分布均匀
3. 货物具有良好的可耐压性，不会由于堆叠而发生变形，且无危险品、特殊物品等货物
4. 所有装载过程认定为离线装载过程
5. 所放置货物的各边与货舱边或集装箱边正交或平行
6. 假设不会出现物资悬空的情况
7. 假设一种货物仅由一种集装箱装载．且货物按照同样的方向，以同样的姿态进行摆放

四、符号说明

符号	含义	单位
x_{ic}	c 号集装箱内第 i 个放置货物的 x 坐标	
y_{ic}	c 号集装箱内第 i 个放置货物的 y 坐标	
z_{ic}	c 号集装箱内第 i 个放置货物的 z 坐标	
m_{ic}	c 号集装箱内第 i 个放置货物的质量	t
v_{ic}	c 号集装箱内第 i 个放置货物的体积	m^3
Z	全体货运飞机的体积利用率	
M_c	c 号货舱的载重最大限制	t
V_c	c 号货舱的体积最大限制	m^3
l_i	第 i 个放置货物的长度	m
w_i	第 i 个放置货物的宽度	m
h_I	第 i 个放置货物的高度	m
L_c	c 号机舱的长度	m
W_c	c 号机舱的宽度	m
H_c	c 号机舱的高度	m
U_{ig}	i 号柔性航空集装箱对于 g 类小型货物的空间利用率	

五、模型建立与求解

5.1 问题一的模型建立与求解

问题一要求我们尽可能地将三架货运飞机填充不留空隙，则问题转化为经典的三维背包装载问题（3D-KLP），同时由于装载货物类型较多且尺寸又各不相同，则进一步认定为强异构类装载问题．其中我们将模型中货物的价值定义为货物的体积，则目标函数转化为货机的体积利用率达到最大，下面开始模型的建立与求解．

5.1.1 目标函数与约束条件的建立

对于本问题，我们定义 C=$\{c_1,c_2...c_9\}$ 为所有可用机舱，从 1-9 分别为大型货机的前中后舱，中型货机的前中后舱，小型货机的前中后舱；定义 G=$\{g_1,g_2,g_3...g_{10}\}$ 为所有可装载货物类别．其次，对于每个长方体货舱，我们在某一角落处进行建系，最长边为 x 轴，其次为 y 轴，最后为 z 轴，对于摆放在其中的货物，我们定义其左后下顶点为其放置的三维坐标点．基于以上分析，对本问题进行模型构建，目标函数与装载约束条件如下：

(1) 由于题目要求尽量使机舱剩余空隙最小，故目标函数设定为集装箱体积利用率最大：

$$Z = \max \frac{\sum_{c=1}^{9}\sum_{i=1}^{n} v_{ic}}{\sum_{c=1}^{9} V_c} \tag{1}$$

其中，Z 为机舱体积利用率,v_{ic} 为 c 号货舱中第 i 个摆放的货物的体积，V_c 为 c 号货舱的体积．

(2) 货物总重量约束：对任一货舱须满足

$$\sum_{i=1}^{n} m_{ic} \leqslant M \quad \forall c \in C \tag{2}$$

其中，等式左侧为该机舱所放置的货物重量之和，等式右侧为该机舱的最大载重限制．

(3) 货物总体积约束：对任一货舱须满足：

$$\sum_{i=1}^{n} v_{ic} \leqslant V \quad \forall c \in C \tag{3}$$

其中，等式左侧为该机舱所放置的货物体积之和，等式右侧为该机舱的货舱体积

(4) 货物装载三维尺寸约束：对任一货舱须满足：

$$\begin{aligned} 0 \leqslant x_i + l_i \leqslant L_c \\ 0 \leqslant y_i + w_i \leqslant W_c \quad \forall c \in C \\ 0 \leqslant z_i + h_i \leqslant H_c \end{aligned} \tag{4}$$

其中，x_i、y_i、z_i 为货物摆放的三维坐标点，l_i、w_i、h_i 为货物的体积参数，L、W、H 为货舱的体积参数．

(5) 货物装载机舱重量平衡约束：为了保证飞行过程平稳，故须保证飞机前中后舱载货量满足最大载货量之比，而由于实际输送过程中不可能完全贴合，故我们设立了 0.1 的误差范围，对任一飞机须满足：

$$\frac{\sum_{i=1}^{n} m_{ic_1}}{\sum_{j=1}^{n} m_{jc_2}} - \frac{M_{c_1}}{M_{c_2}} \leqslant 0.1 \quad \forall c_1, c_2 \in C \tag{5}$$

其中 c_1,c_2 取自同一架飞机的货舱.

5.1.2 利用三空间分割规则进行货物摆放

下面我们进行装舱约束，对于每个货舱长方体，我们首先制定了如下的装舱规则：

(1) 定序规则

货物装载顺序将会影响集装箱空间布局的质量，在设计装载方案的过程中，为了确定货物放入集装箱中的先后顺序，较多地采用定序规则．在此规则中常用的有体积递减、重量递减、最长边递减等规则．在此问题中，我们按照货物密度对货物进行升序排序，优先考虑摆放密度最小的货物.

(2) 定位规则

在待装载货物遵循一定的先后次序逐个装箱的过程中，需要相应的定位规则确定其摆放位置．定位规则包含多种方法，首先将货物放置在集中的一角，然后沿着边依次装载，最后再填满中心的占角策略应用较多．本文采用占角策略开展装载研究.

(3) 三空间分割规则

首先我们对空间进行编码，定义空间 S=(x_a,y_a,z_a,x_b,y_b,z_b), 其中前三个坐标表示空间的左后下顶点，后三个坐标表示空间的右前上顶点，所以空间的长度 L，宽度 W，高度 H 表示为：

$$\begin{aligned} L &= \max(x_b - x_a, y_b - y_a) \\ W &= \min(x_b - x_a, y_b - y_a) \\ H &= z_b - z_a \end{aligned} \tag{6}$$

定义货舱的剩余空间指装载一定的货物以后，货舱内部剩余的还可以利用的空间．在后续不断添加货物的过程中，剩余空间的形状变得相对复杂，很难进行整体描述．为了使货物的待装载空间表述清楚，采用三空间分割规则来对集装箱空间进行划分．当货舱中装入一个货物时，此时货舱中的空间结构发生了变化，除去货物所占有的体积外，将剩余空间划分成前、右、上三个部分．在不断添加货物时，陆续生成上述子空间，直至装满集装箱或者没有待装货物为止．集装箱三空间分割布局与三维坐标系对应关系示意图如图 1.

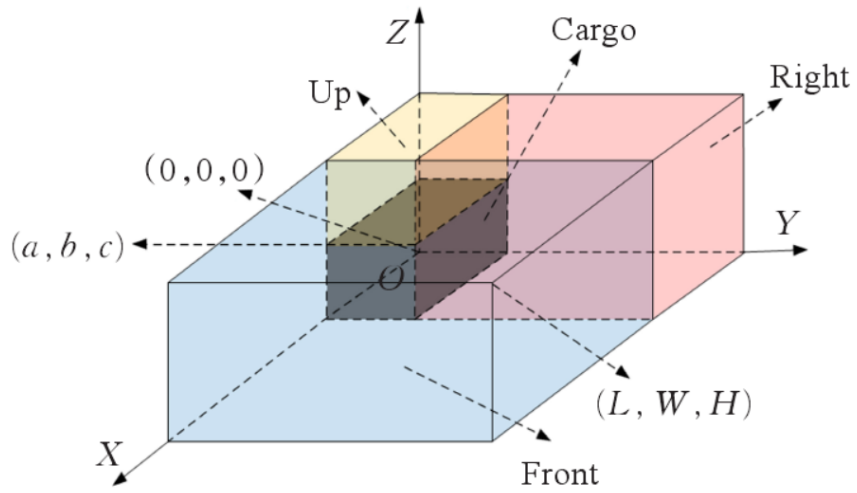


图 1 三空间分割剩余空间示意图

在每次放置货物后进行三空间分割，然后在每个子空间中进行下一次的摆放，同样沿用占角策略，此时我们对十种货物进行搜索，若满足以下约束条件以及 5.1.1 中模型建立的约束条件，则可进行放置.

$$\begin{aligned} l_i &\leqslant L \\ w_i &\leqslant W \\ h_i &\leqslant H \end{aligned} \tag{7}$$

假设在放置货物 $g_i(l_i,w_i,h_i)$ 后，原空间 $V_i(x_{ia},y_{ia},z_{ia},x_{ib)},y_{ib},H)$ 被分为以下三个子空间，前 V_{i_1}，右 V_{i_2}，后 V_{i_3}，其分别对应的空间编码如下：

$$\begin{aligned} &V_{i_1}(x_{ia}+l_i, y_{ia}, z_{ia}, x_{ib}, y_{ib}, H) \\ &V_{i_2}(x_{ia}, y_{ia}+k_i, z_{ia}, x_{ia}+l_i, y_{ib}, H) \\ &V_{i_3}(x_{ia}, y_{ia}, z_{ia}+h_i, x_{ia}+l_i, y_{ia}+w_i, H) \end{aligned} \tag{8}$$

至此模型建立完毕，下面开始模型的求解，我们利用 python 进行编程寻找最优的货运方案，总求解过程的流程图如图 2 所示，具体代码见附录一，得出货运结果如下：

表 1 货舱运送设计表

	HW3	HW10	HW1	HW9	HW6	载重利用率	体积利用率
货舱 1	7	4	1	2	0	1.0000	0.0291
货舱 2	7	4	4	1	2	1.0000	0.0322
货舱 3	7	4	0	2	0	0.9875	0.0303
货舱 4	7	4	0	2	0	0.9875	0.0616
货舱 5	7	4	2	1	2	0.9834	0.0495
货舱 6	7	3	0	0	0	0.9667	0.0643
货舱 8	7	4	0	2	0	0.9875	0.3675
货舱 9	5	1	0	0	0	0.9500	0.2848

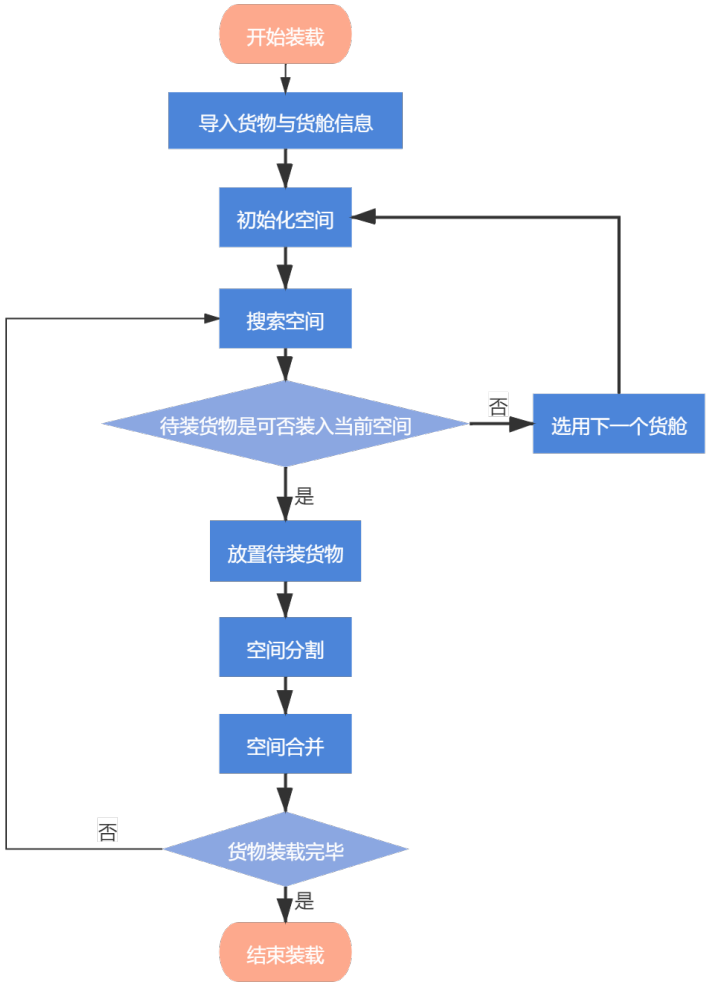


图 2 货运方案算法求解流程图

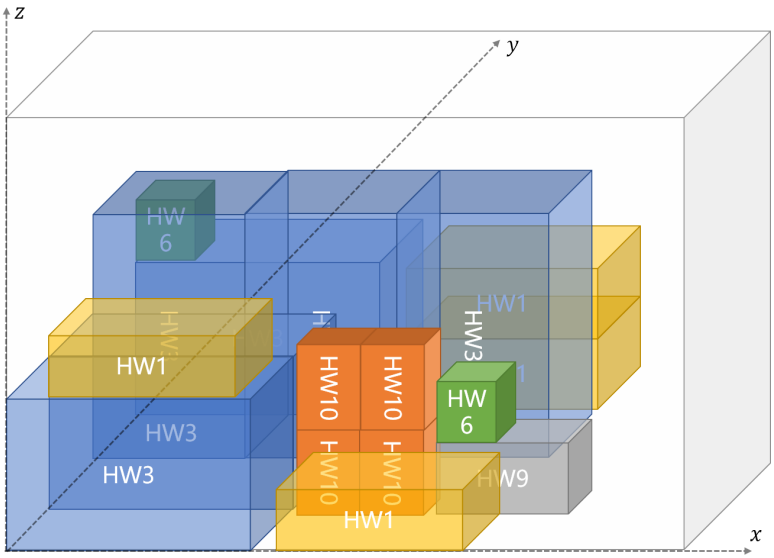


图 3 大型飞机中舱的货运设计示意图

同时我们对结果进行可视化，部分结果如图 3.

5.2 问题二的模型建立与求解

对于问题二，我们沿袭第一问的思路，选择将集装箱和货舱的空间利用率作为目标函数，建立模型求解目标函数的最大值，并得出相应的摆放情况，以及所需的飞机架次．我们从实际情况出发，先保证集装箱的利用率进行选定集装箱，再考虑机舱的空间利用率从而确定机型．由于飞机架次不受限制，故我们将第一问的模型进行改进，着重体现在改变飞机的定序规则，求解最优空间利用率下的定序规则，机型选择与飞机所需的架次．

5.2.1 选择集装箱并进行集装箱装配

我们首先进行初步考量，对于大型货机，用货物的总重量除以大型货机的总载重量，得到大货机至少需要 6299.53/34≈185 架次，数值较为庞大，对于中、小货机，同样计算出也至少需要 242，350 架．其次我们考虑到由于集装箱非一次性用品，可以反复使用，并且货机架次多，单次飞行所需集装箱数量较少，故我们不考虑集装箱的个数限制，仅考虑集装箱的体积限制．

(1) 集装箱的初步选择：

我们观察到：仅有货物 HW2、HW6、HW7、HW10 体积小于 2m³，集装箱 Y1、Y2、Y4、Y5 外部尺寸相等，Y3、Y6 外部尺寸极其相近，而 Y5、Y6 为柔性集装箱，其高度可变，可对无用体积进行高度的压缩，对于增大集装箱的体积利用率有较大作用，并且其质量相对较轻，所以我们用 Y5、Y6 柔性航空集装箱对 Y1、Y2、Y3、Y4 航空集装箱进行替代，并且同时考虑具有独特尺寸的 Y7 柔性航空集装箱，故最终决定使用 Y5、Y6、Y7 三种柔性航空集装箱装载体积小于 2m³ 的货物．

(2) 建立同构同向货物的二维装箱模型，进行集装箱的再选择：

由于所选集装箱均为柔性航空集装箱，其高度可变，故对于集装箱的体积利用率，我们可以将其转化为底面积的利用率．我们选取底面积利用率最高的集装箱作为该种货物装载的集装箱．我们定义第 i 种柔性航空集装箱对于第 j 种货物的空间最大利用率为

$$U_{ij} = \max \frac{a_j b_j \lfloor \frac{L_i}{a_j} \rfloor \lfloor \frac{W_i}{b_j} \rfloor}{L_i W_i} \tag{9}$$

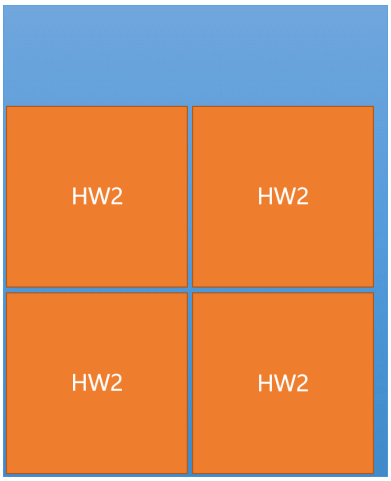
其中 $i \in \{1,2,3\}$，$j \in \{1,2,3,4\}$，a_j,b_j 是第 j 种货物长宽高 l_j,w_j,h_j 的所有可能组合．

对每一类小型货物，我们我们通过计算 U_{1j}、U_{2j}、U_{3j}，选取其中的最大值，并将其集装箱型号和货物摆放方式记录，作为第 j 种货物填充所用的集装箱和第 j 种货物的填充状态．

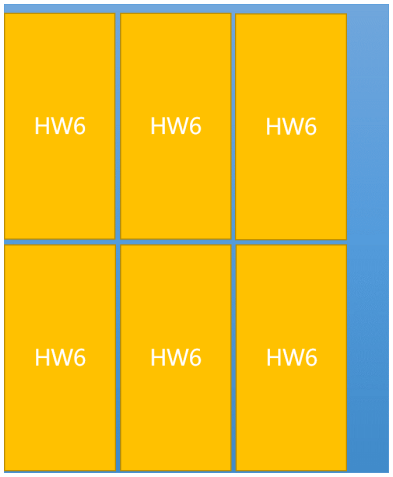
我们编写相应代码进行求解得出下表结果，具体代码见附录 2，最终决定四种小型货物均采用 Y7 柔性集装箱进行装载，具体摆放的俯视图如图 4 所示．

表 2 集装箱选择比较表

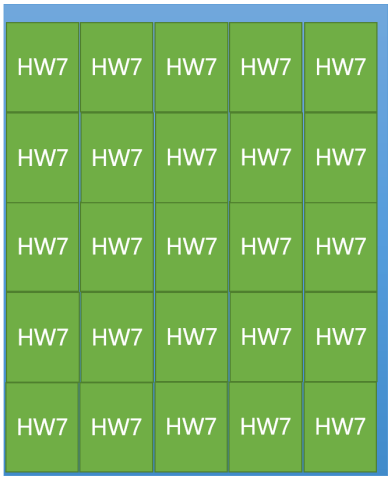
U	HW2	HW6	HW7	HW10
Y5 柔性集装箱	0.6524	0.7499	0.8078	0.5918
Y6 柔性集装箱	0.6935	0.6763	0.8242	0.629
Y7 柔性集装箱	0.7815	0.8983	0.9679	0.7089



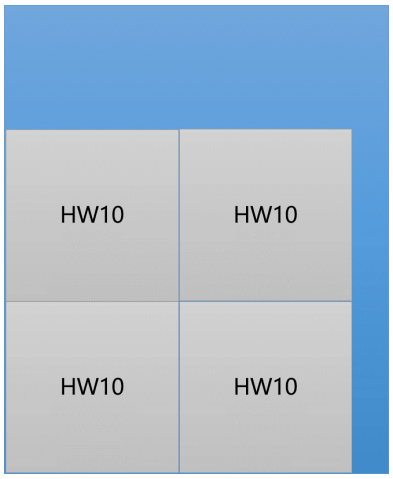
(a) Y7 集装箱内 HW2 摆放示意图



(b) Y7 集装箱内 HW6 摆放示意图



(c) Y7 集装箱内 HW7 摆放示意图



(d) Y7 集装箱内 HW10 摆放示意图

图 4 集装箱装箱示意图

(3) 对货物数据进行更新:

我们根据集装箱和填充状态，按照高度的限制，更新体积小于 $2m^3$ 的货物的数据，将十类货物中的小型货物用 Y7 柔性集装箱所替代，具体更新其长度，宽度，高度，重量，供应数量等参数，其中长度均为 2.235m，宽度均为 2.743m，通过计算更新数据如下表：

表 3 集装箱装配设计表

小型货物	装载数量	集装箱高度	集装箱总载重量	集装箱的空间利用率	集装箱的需求量
HW2	4	1.15	0.95t	78.15%	92
HW6	12	1.82	3.75t	89.83%	26
HW7	25	1.08	5.9t	96.76%	24
HW10	4	1.6	0.95t	70.89%	306

5.2.2 选择货机机型确定具体配送方案

在进行完集装箱装配后，我们开始进行装机过程，我们对第一问中的模型进行改进，增加了货物的数量约束条件，保证货物可以全部装机．其次改变定序规则，我们分别从按密度递增，按体积递减，按最长边递减三种定序规则对三种机型分别进行了求解，最终选定空间利用率最大的选择作为最终的机型选择与摆放方案．我们利用 python 进行求解，具体代码见附录 3，得出下表结果：

表 4 机型与定义规则选择表

定序规则	机型	所需架次	货机空间体积利用率
密度递增	大型货机	194	0.015472
	中型货机	305	0.021586
	小型货机	375	0.137183
体积递减	大型货机	193	0.015564
	中型货机	303	0.021689
	小型货机	372	0.138449
最长边递减	大型货机	192	0.015640
	中型货机	299	0.021981
	小型货机	370	0.139084

根据题目要求，为使货舱尽量不留空隙，则要求空间体积利用率最大，故选择小型货机进行配送，需要 370 架次飞机，具体运送方案见支撑材料 small_cargo_length.xlsx. 表中 J2,J6,J7,J10 即代表运送 HW2,HW6,HW7,HW10 货物的集装箱，自上而下为按前中后舱排列的装配方案．

5.3 问题三的模型建立与求解

通过对大中小飞机体积利用率的比较，我们发现小飞机在体积利用率上明显高于大中型飞机，若仍以集装箱尽量不留空隙、货运飞机尽量不留空隙作为货物装运的思路，即以空间利用率尽可能高作为货物装运的思路，不考虑飞机的其它成本，确实会影响经济效益．我们考虑到在实际生活中，货物运输的利润不仅仅是只依靠于货物本身的价值，同样也取决于交通运输的成本．已知整个货运过程的利润 Q 等于货物总价值减去总成本 K，总成本又包括货物置备成本、货物的空中运输成本和由于装卸货，飞机起降带来的飞机的附加成本．由于货

源已经确定，货物数量也不再发生变化，即货物总价值 Value、货物置备成本、货物的运输成本不发生改变，所以如果仅仅只考虑空间利用率不足以获得经济效益最大．因此在第三问中我们引入三种机型的附加成本 A、B、C，来达到提高经济效益的目的，即：

$$\begin{aligned} Q &= Value_{货物} - K \\ K &= K_{货物} + K_{运输} + K_{附加} \end{aligned} \tag{10}$$

示意图如下：

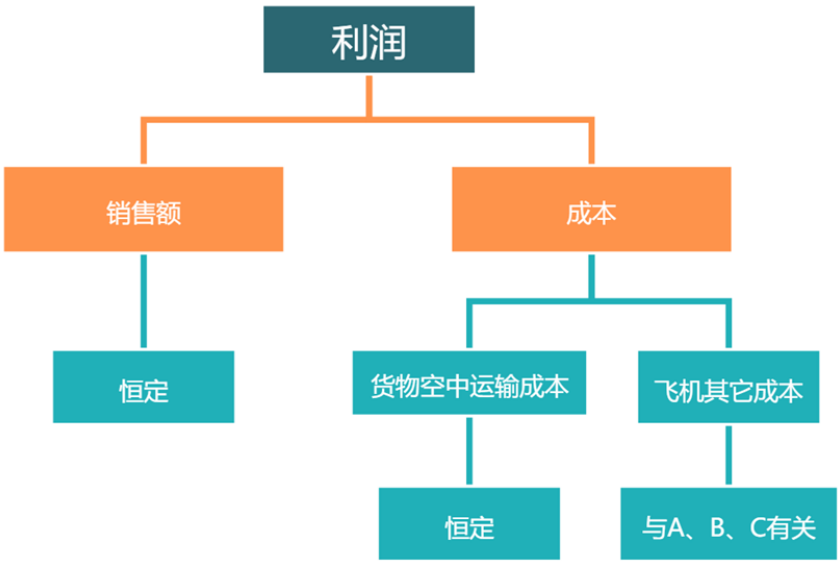


图 5 货运利润利润图

为了更好地建立模型同时避免将问题复杂化，我们补充以下假设：

1. 货物的数量不发生改变，且其经济效益能够发挥到最大
2. 每架大、中、小三种类型的飞机的其它成本分别是 A、B、C（万元）且恒定不变
3. 飞机的飞行架次没有限制
4. 货物的运输只选定一种类型的飞机

为了降低成本，我们则要尽量减少飞机的架次，由表 4 可得，三种机型均在采用最长边递减的定序规则下所需架次最少，分别为 192，299，370 架次．则若要利润最大，就要选择成本最小的货运方案，那么对于大、中、小三种同类型的货机，在三种定序规则下所产生的最小的飞机附加成本分别为 192A，299B，370C．则货机运输附加成本最小值 $K_{附加}=\min(192A, 299B, 370C)$，同时我们计算出货物的销售总利润 $Value - K_{货物} = 1550$ 万元，则最大利润

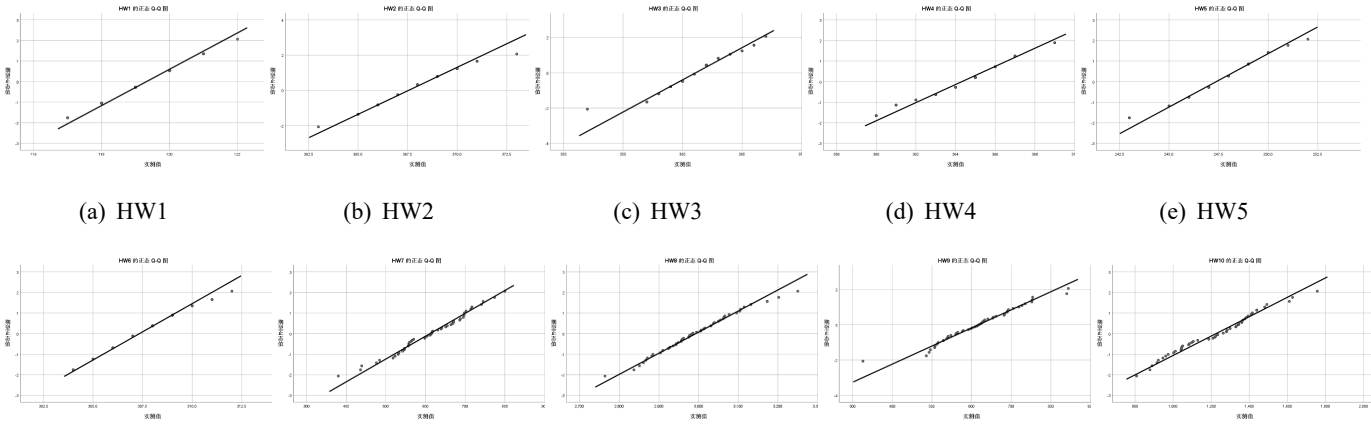
$$Q = \max(1550 - 192A, 1550 - 299B, 1550 - 370C) \tag{11}$$

相应地根据不同 Q 的取值选择大、中、小飞机进行运输，均按照最长边递减的定序规则进行装配，具体的货物摆放设计表见支撑文件 big_cargo_length.xlsx,middle_cargo_length.xlsx,small_cargo_length.xlsx.

5.4 问题四的模型建立与求解

5.4.1 对问题中可靠性的定义

我们观察附件 2 所给货物历史数据，发现货物的历史需求量可能会符合某种分布规律，于是我们画出了每种货物历史需求量的 Q-Q 图如图 6，发现每种货物的历史需求量所代表的数据点基本都分布在一条直线上，所以我们认定每种货物的历史需求量近似符合正态分布.



(a) HW1 (b) HW2 (c) HW3 (d) HW4 (e) HW5

(f) HW6 (g) HW7 (h) HW8 (i) HW9 (j) HW10

图 6 各类货物历史需求量正态检验 Q-Q 图

其次根据正态分布的 3σ 原则，数值分布在 $(\mu-\sigma,\mu+\sigma)$ 中的概率为 0.6827，数值分布在 $(\mu-2\sigma,\mu+2\sigma)$ 中的概率为 0.9545，我们将第四问中可靠性 95% 与概率 0.9545 相关联，将第五问中可靠性 70% 与 0.6827 相关联，进而给出可靠性的定义：可靠性为 95% 是指：附件二中各种货物历史数据仅有 95% 的数据可代表实际的运输情形，即实际的运输时的每种货物需求量，只能取自在 $(\mu-2\sigma,\mu+2\sigma)$ 的历史数据.

5.4.2 在可靠性为 95% 下求解最大利润货运方案

为方便计算，我们将每种货物的实际需求量离散化，假定为三种数值，分别为 $\mu_i-2\sigma_i$，μ_i，$\mu_i+2\sigma_i(i$ 为货物种类)，设定其概率分布分别为：

$$\begin{aligned} P(\mu-2\sigma)&=\frac{P(\mu-2\sigma,\mu-\sigma)}{P(\mu-2\sigma,\mu+2\sigma)}=\frac{0.1359}{0.9544}=0.1424\\ P(\mu)&=\frac{P(\mu-\sigma,\mu+\sigma)}{P(\mu-2\sigma,\mu+2\sigma)}=\frac{0.6826}{0.9544}=0.7152\\ P(\mu+2\sigma)&=\frac{P(\mu+\sigma,\mu+2\sigma)}{P(\mu-2\sigma,\mu+2\sigma)}=\frac{0.1359}{0.9544}=0.1424 \end{aligned} \tag{12}$$

其中的概率是指历史数据的分布概率．同时，我们将所有货物的装运总量也只分为三种情况：情况 A：每种货物均只装 $\mu_i-2\sigma_i$ 件，情况 B：每种货物均只装 μ_i 件，情况 C：每种货物均只装 $\mu_i+2\sigma_i$ 件，根据公式计算利润期望，绘制表格以及图像.

我们首先套用第三问的模型对每种货物量取值针对不同机型分别求解出其所需的机架数，结果如下表

表 5 不同载货量选择的机架需求表

	A:$\mu-2\sigma$	B:μ	C:$\mu+2\sigma$
大型货机	175	192	210
中型货机	271	299	327
小型货机	336	370	405

我们考虑货物的出售情况，定义货物所产生的净利润的期望值为：

$$Q_{\text{净利润}} = P(\mu-2\sigma)Q_1 + P(\mu)Q_2 + P(\mu+2\sigma)Q_3) \tag{13}$$

$$Q_i = \text{货物正常销售额} + \text{货物甩卖额-货物运输价格-货物装机前成本-飞机的附加成本} \tag{14}$$

$$\text{货物甩卖额} = \text{货物运输价格的30\%} \tag{15}$$

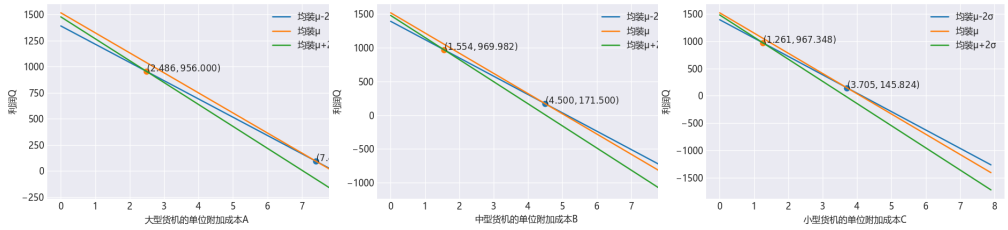
$$\text{货物装机前成本} = \text{货物运输价格的40\%} \tag{16}$$

其中 Q_1,Q_2,Q_3 分别表示在当前类别货物装载时，实际货物需求量分别为为 $\mu_i-2\sigma_i$，μ_i，$\mu_i+2\sigma_i$ 的利润，计算求解得三种载货情况的净利润分别为 1391，1517，1478 万元，进而我们得出了不同机型不同装货方案下的利润如下表所示

表 6 不同载重量的利润期望表

	均装 μ-2σ	均装 μ	均装 μ+2σ
大型货机	1391-175A	1517-192A	1478-210A
中型货机	1391-271B	1517-299B	1478-327B
小型货机	1391-336C	1517-370C	1478-405C

进而我们分别绘制出三种机型利润随单位附加成本变化的曲线，如下图所示：



(a) 大型货机　　(b) 中型货机　　(c) 小型货机

图 7 不同机型利润期望随单位附加成本的变化示意图

所以最终的货运装运策略取决于各机型单位附加成本的取值，选取利润最高的机型及装配方案即可，其中装配时定序规则依然采用按最长边递减的规则，对应的九种情况的货机详细摆放设计见支撑材料.

5.5 问题五的模型建立与求解

5.5.1 对问题中可靠性的定义

我们依据第四问原理，同理将可靠性 70%与 0.6827 相关联，给出可靠性的定义，可靠性为 70%是指：附件二中各种货物历史数据仅有 70%的数据可代表实际的运输情形，即实际的运输时的每种货物需求量，只能取自 $(\mu-\sigma,\mu+\sigma)$ 的历史数据.

5.5.2 在可靠性为 70% 下求解最大利润货运方案

同第四问，为方便计算，我们将每种货物的实际需求量离散化，假定为三种数值，分别为 $\mu_i-\sigma_i$，μ_i，$\mu_i+\sigma_i(i$ 为货物种类)，设定其概率分布分别为：

$$\begin{aligned} P(\mu-\sigma)&=\frac{P(\mu-\sigma,\mu-\sigma/2)}{P(\mu-\sigma,\mu+\sigma)}=\frac{0.1498}{0.6826}=0.2195\\ P(\mu)&=\frac{P(\mu-\sigma/2,\mu+\sigma/2)}{P(\mu-\sigma,\mu+\sigma)}=\frac{0.3830}{0.6826}=0.5610\\ P(\mu+\sigma)&=\frac{P(\mu+\sigma/2,\mu+\sigma)}{P(\mu-\sigma,\mu+\sigma)}=\frac{0.1489}{0.6826}=0.2195 \end{aligned} \tag{17}$$

其中的概率是指历史数据的分布概率．同时，我们将所有货物的装运总量也只分为三种情况：情况 A：每种货物均只装 $\mu_i-\sigma_i$ 件，情况 B：每种货物均只装 μ_i 件，情况 C：每种货物均只装 $\mu_i+\sigma_i$ 件，根据公式计算利润期望，绘制表格以及图像．我们同样套用第三问的模型对每种货物量取值针对不同机型分别求解出其所需的机架数，结果如下表

表 7 不同载货量选择的机架需求表

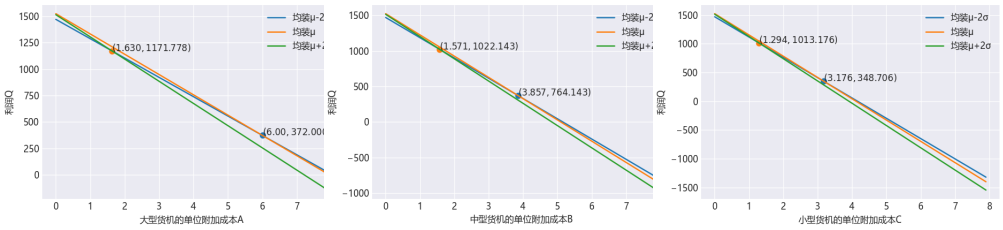
	A:$\mu-\sigma$	B:μ	C:$\mu+\sigma$
大型货机	183	192	201
中型货机	285	299	313
小型货机	353	370	387

我们考虑货物的出售情况，根据 (13)(14)(15)(16) 计算求解得三种载货情况的净利润分别为 1470，1524，1514 万元，进而我们得出了不同机型不同装货方案下的利润如下表所示

表 8 不同载重量的利润期望表

	均装 μ-2σ	均装 μ	均装 μ+2σ
大型货机	1470-183A	1524-192A	1514-201A
中型货机	1391-285B	1524-299B	1514-327B
小型货机	1391-353C	1524-370C	1514-387C

进而我们分别绘制出三种机型利润随单位附加成本变化的曲线，如下图所示：



(a) 大型货机　　(b) 中型货机　　(c) 小型货机

图 8 不同机型利润期望随单位附加成本的变化示意图

所以最终的货运装运策略取决于各机型单位附加成本的取值，选取利润最高的机型及装配方案即可，其中装配时定序规则依然采用按最长边递减的规则，对应的九种情况的货机详细摆放设计见支撑材料.

六、模型的评价与改进

6.1 模型的优点

- （我们在货物装配过程中，首先通过定位规则和定序规则确定了装配的顺序，同时以求解最大空间利用率为目标函数，通过三空间切割原则来保证装载的有序进行. 在多个条件的约束下，能够寻找到空间利用率的最优解，避免的了“组合爆炸”现象的发生.
- 在寻求最大经济效益的过程中，我们不断变换定序规则，通过密度升序、体积降序、最长边降序来得到同种机型下的最优解，从而使结果更优化.
- 在第四、五问中，我们将可靠性与数据的正态性进行联系，数据得到较好的拟合，并通过概率论对多种情况的最终经济效益进行分析，比较好的贴近实际.

6.2 模型的缺点

- 在三空间切割过程中，由于不断的递归迭代，可能会出现数据的紊乱或者细节性的误差，这一点我们并未有效解决.
- 在第三、四、五问中，我们未能对货物的需求以及公司可能的风险进行更加精准的预测，只能根据已有的数据，在众多约束条件中求得未来较短时间内的估计，时间的长远性有待改进.

6.3 模型的改进

- 在应用三空间切割原则进行空间分割与合并时，可尝试将每个子空间独立出来，这样可以优化算法的时间复杂度以及避免了递归调用时的细节误差.
- 在对风险及货物的需求量进行预估分析时，我们可以应用时间序列分析的方法，从而更加精准地计算出货物的需求量，以便更好地达到最大的经济效益以及最优化设计装配方案.

参考文献

[1] 张钧, 贺可太. 求解三维装箱问题的混合遗传模拟退火算法 [J]. 计算机工程与应用,2019,55(14):32-39+47.

[2] 张德富, 魏丽军, 陈青山, 陈火旺. 三维装箱问题的组合启发式算法 [J]. 软件学报,2007(09):2083-2089.

[3] Romão, Oberlan Christo, André Gustavo dos Santos, and José Elias Claudio Arroyo. ”A two phase hybrid method to support 2D and 3D vehicle loading.” 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA). IEEE, 2012.

[4] Ramos, A. Galrão, et al. ”A container loading algorithm with static mechanical equilibrium stability constraints.” Transportation Research Part B: Methodological 91 (2016): 565-581.

附 录

附录 1：问题一求解货运装配方案代码

```
import pandas as pd
import matplotlib.pyplot as plt
import math
import numpy as np
def load(m): #对第m个机开始装配，m：0-8
    Space=[0,0,0,data.L[m],data.W[m],data.H[m]]
    for location in S:
        if location[3]==1:
            front_load(m,location,Space)
        elif location[3]==2:
            #right_load(m,2.96,location,Space)
            right_load(m,3.17,location,Space)
        elif location[3]==3:
            #up_load(m,2.96,2.12,location,Space)
            up_load(m,3.17,0.85,location,Space)
def constraint1(m,i): #在m号舱每个可放置点放置i时判断约束条件是否满足约束条件
    if sum(solution.mim[m])+data.m[i] > data.M[m]: #重量约束
        return False
    if sum(solution.vim[m])+data.v[i] > data.V[m]: #体积约束
        return False
    return True
def constraint2(m,i,location): #三维尺寸约束
    if location[0]+data.l[i]>data.L[m] or location[1]+data.w[i]>data.W[m] or location
        [2]+data.h[i]>data.H[m]:
        return False
    return True
def front_load(m,location,space):
    space[0]+=location[0]
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否前空间成功放置
    for i in range(10):
        choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
        for j in choice:
            for k in choice:
                for l in choice:
                    if k!=j and k!=l:
                        if j<L_space and k<W_space and l<H_space:
                            if constraint1(m,i)==True and constraint2(m,i,location)==True:
                                solution.Cim[m].append(i)
                                solution.nameim[m].append(data.name[i])
```

```
                            solution.xim[m].append(j)
                            solution.yim[m].append(k)
                            solution.zim[m].append(l)
                            solution.vim[m].append(data.v[i])
                            solution.mim[m].append(data.m[i])
                            #S.remove(location)
                            flag=1
                            break
    if flag==1:
        location1=[location[0]+j,0,0,1]
        location2=[location[0],k,0,2]
        location3=[location[0],0,l,3]
        front_load(m,location1,space)
        right_load(m,j,location2,space)
        up_load(m,j,k,location3,space)
def right_load(m,var,location,space):
    space[1]+=location[1]
    space[3]=location[0]+var
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否右空间成功放置
    for i in range(10):
        choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
        for j in choice:
            for k in choice:
                for l in choice:
                    if k!=j and k!=l:
                        if j<L_space and k<W_space and l<H_space:
                            if constraint1(m,i)==True and constraint2(m,i,location)==True:
                                solution.Cim[m].append(i)
                                solution.nameim[m].append(data.name[i])
                                solution.xim[m].append(j)
                                solution.yim[m].append(k)
                                solution.zim[m].append(l)
                                solution.vim[m].append(data.v[i])
                                solution.mim[m].append(data.m[i])
                                #S.remove(location)
                                flag=1
                                break
    if flag==1:
        location1=[j,0,0,1]
        location2=[0,location[1]+k,0,2]
        location3=[0,location[1],l,3]
        front_load(m,location1,space)
        right_load(m,j,location2,space)
```

```
        up_load(m,j,k,location3,space)
def up_load(m,var1,var2,location,space):
    space[2]+=location[2]
    space[3]=location[0]+var1
    space[4]=location[1]+var2
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否右空间成功放置
    for i in range(10):
        choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
        for j in choice:
            for k in choice:
                for l in choice:
                    if k!=j and k!=l:
                        if j<L_space and k<W_space and l<H_space:
                            if constraint1(m,i)==True and constraint2(m,i,location)==True:
                                solution.Cim[m].append(i)
                                solution.nameim[m].append(data.name[i])
                                solution.xim[m].append(j)
                                solution.yim[m].append(k)
                                solution.zim[m].append(l)
                                solution.vim[m].append(data.v[i])
                                solution.mim[m].append(data.m[i])
                                #S.remove(location)
                                flag=1
                                break
    if flag==1:
        location1=[j,0,location[2],1]
        location2=[0,k,location[2],2]
        location3=[0,0,location[2]+l,3]
        front_load(m,location1,space)
        right_load(m,j,location2,space)
        up_load(m,j,k,location3,space)
class Data:
    #九个货舱 大前大中大后中前中中中后小前小中小后
    L=[18,20,16,15,18,13,10,12,9] #货机的长
    W=[8.45,8.45,8.45,5.48,5.48,5.48,3.45,3.45,3.45] #货机的宽
    H=[13.4,14.8,12.6,10.2,13.4,9.7,3.4,3.4,3.4] #货机的高
    V=[2038.14,2501.2,1703.52,838.44,1321.776,691.028,117.3,140.76,105.57]#货机体积
    M=[10,16,8,8,12,6,6,8,4]#货机载重量
    #十种货物
    l=list(df2['长度']) #货物长度
    w=list(df2['宽度']) #货物宽度
    h=list(df2['高度']) #货物高度
    v=list(df2['货物体积']) #货物体积
```

```
    #rou=list(df2['货物密度']) #货物密度 kg/m3
    m=list(df2['重量']) #货物重量
    N=list(df2['平均供应件数']) #供货数量
    name=list(df2['货物名称']) #货物名称
data=Data()
class Solution:
    Cim=[[]for i in range(9)] #存储第i个货物的类别
    xim=[[]for i in range(9)] #存储第m个货机中第i个放置的货物的x坐标
    yim=[[]for i in range(9)] #存储y坐标
    zim=[[]for i in range(9)] #存储z坐标
    mim=[[]for i in range(9)] #存储第i个货物的重量
    vim=[[]for i in range(9)] #存储第i个货物的体积
    nameim=[[]for i in range(9)] #第i个货物的类别名
solution=Solution()
S=[] #可放置点
#S.append((2.96,0,0,1))
#S.append((0,0,1.21,3))
#S.append((0,2.12,0,2))
S.append((3.17,0,0,1))
S.append((0,0.85,0,2))
S.append((0,0,2.12,3))
for i in range(9): #都装上第一个货物
    solution.nameim[i].append(data.name[0])
    solution.xim[i].append(0)
    solution.yim[i].append(0)
    solution.zim[i].append(0)
    #solution.mim[i].append(2.1)
    solution.mim[i].append(0.7)
    #solution.vim[i].append(7.592992)
    solution.vim[i].append(5.712340)
    solution.Cim[i].append(0)
data.N[0]-=9
for i in range(9):
    load(i)
```

附录 2：问题二求解集装箱装配方案代码

```
def space(my_list, L, W):
    alist = []
    for i in my_list:
        for j in my_list:
            if i == j:
                continue
            else:
                U = (i * j * (L // i) * (W // j))/(L * W)
```

```
                #print(i, j)
                alist.append(U)
    return alist
my_list = [[1.32, 0.64, 0.84], [0.98, 0.42, 0.52], [1.5, 1]]
for i in my_list:
    x = space(i, 2.891, 2.338)
    x.sort()
    print(x.pop())
print()
for i in my_list:
    y = space(i, 1.36, 2.338)
    y.sort()
    print(y.pop())
print()
for i in my_list:
    z = space(i, 2.235, 2.643)
    z.sort()
    print(z.pop())
```

附录 3：问题二求解货运装配方案代码

```
import pandas as pd
import matplotlib.pyplot as plt
import math
import numpy as np
df1=pd.read_csv("D:/desktop/1.csv")[:9]
df2=pd.read_csv("D:/desktop/2.csv")[:10]
class Data:
    #九个货舱 大前大中大后中前中中中后小前小中小后
    #中飞机
    L=[15,18,13]*2000 #货机的长
    W=[5.48,5.48,5.48]*2000 #货机的宽
    H=[10.2,13.4,9.7]*2000 #货机的高
    V=[838.44,1321.776,691.028]*2000#货舱体积
    M=[8,12,6,6,8,4]*2000#货机载重量
    #大飞机
    L=[18,20,16]*2000
    W=[8.45,8.45,8.45]*2000
    H=[13.4,14.8,12.6]*2000
    V=[2038.14,2501.2,1703.52]*2000
    M=[10,16,8]*2000
    #小飞机
    L=[10,12,9]*2000
    W=[3.45,3.45,3.45]*2000
    H=[3.4,3.4,3.4]*2000
```

```
    V=[117.3,140.76,105.57]*2000
    M=[6,8,4]*2000
    #十种货物
    l=list(df2['长度']) #货物长度
    w=list(df2['宽度']) #货物宽度
    h=list(df2['高度']) #货物高度
    v=list(df2['货物体积']) #货物体积
    #rou=list(df2['货物密度']) #货物密度 kg/m3
    m=list(df2['重量']) #货物重量
    N=list(df2['平均供应件数']) #供货数量
    name=list(df2['货物名称']) #货物名称
data=Data()
class Solution:
    Cim=[[]for i in range(2000)] #存储第i个货物的类别
    xim=[[]for i in range(2000)] #存储第m个货机中第i个放置的货物的x坐标
    yim=[[]for i in range(2000)] #存储y坐标
    zim=[[]for i in range(2000)] #存储z坐标
    mim=[[]for i in range(2000)] #存储第i个货物的重量
    vim=[[]for i in range(2000)] #存储第i个货物的体积
    nameim=[[]for i in range(2000)] #第i个货物的类别名
solution=Solution()
def load(m): #对第m个机开始装配
    Space=[0,0,0,data.L[0],data.W[m],data.H[m]]
    S=[] #可放置点
    flag2=0
    for num in range(10):
        if data.N[num]>0:
            solution.nameim[m].append(data.name[num])
            solution.xim[m].append(0)
            solution.yim[m].append(0)
            solution.zim[m].append(0)
            solution.mim[m].append(data.m[num])
            solution.vim[m].append(data.v[num])
            solution.Cim[m].append(num)
            data.N[num]-=1
            S.append((data.l[num],0,0,1))
            S.append((0,data.w[num],0,2))
            S.append((0,0,data.h[num],3))
            flag2=1
            break
    if flag2==1:
        for location in S:
            if location[3]==1:
                front_load(m,location,Space)
            elif location[3]==2:
                right_load(m,location[0],location,Space)
```

```
            elif location[3]==3:
                up_load(m,location[0],location[1],location,Space)
def constraint1(m,i): #在m号舱每个可放置点放置i时判断约束条件是否满足约束条件
    if sum(solution.mim[m])+data.m[i] > data.M[m]: #重量约束
        return False
    if sum(solution.vim[m])+data.v[i] > data.V[m]: #体积约束
        return False
    return True
def constraint2(m,i,location): #三维尺寸约束
    if location[0]+data.l[i]>data.L[m] or location[1]+data.w[i]>data.W[m] or location
        [2]+data.h[i]>data.H[m]:
        return False
    return True
def front_load(m,location,space):
    space[0]+=location[0]
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否前空间成功放置
    for i in range(10):
        if data.N[i]>0:
            choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
            for j in choice:
                for k in choice:
                    for l in choice:
                        if k!=j and k!=l:
                            if j<L_space and k<W_space and l<H_space:
                                if constraint1(m,i)==True and constraint2(m,i,location)==
                                    True:
                                    solution.Cim[m].append(i)
                                    solution.nameim[m].append(data.name[i])
                                    solution.xim[m].append(j)
                                    solution.yim[m].append(k)
                                    solution.zim[m].append(l)
                                    solution.vim[m].append(data.v[i])
                                    solution.mim[m].append(data.m[i])
                                    data.N[i]-=1
                                    #S.remove(location)
                                    flag=1
                                    break
    if flag==1:
        location1=[location[0]+j,0,0,1]
        location2=[location[0],k,0,2]
        location3=[location[0],0,l,3]
        front_load(m,location1,space)
        right_load(m,j,location2,space)
```

```
        up_load(m,j,k,location3,space)
def right_load(m,var,location,space):
    space[1]+=location[1]
    space[3]=location[0]+var
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否右空间成功放置
    for i in range(10):
        choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
        if data.N[i]>0:
            for j in choice:
                for k in choice:
                    for l in choice:
                        if k!=j and k!=l:
                            if j<L_space and k<W_space and l<H_space:
                                if constraint1(m,i)==True and constraint2(m,i,location)==
                                    True:
                                    solution.Cim[m].append(i)
                                    solution.nameim[m].append(data.name[i])
                                    solution.xim[m].append(j)
                                    solution.yim[m].append(k)
                                    solution.zim[m].append(l)
                                    solution.vim[m].append(data.v[i])
                                    solution.mim[m].append(data.m[i])
                                    data.N[i]-=1
                                    #S.remove(location)
                                    flag=1
                                    break
    if flag==1:
        location1=[j,0,0,1]
        location2=[0,location[1]+k,0,2]
        location3=[0,location[1],l,3]
        front_load(m,location1,space)
        right_load(m,j,location2,space)
        up_load(m,j,k,location3,space)
def up_load(m,var1,var2,location,space):
    space[2]+=location[2]
    space[3]=location[0]+var1
    space[4]=location[1]+var2
    L_space=max(space[3]-space[0],space[4]-space[1])
    W_space=min(space[3]-space[0],space[4]-space[1])
    H_space=space[5]
    flag=0 #标志是否上空间成功放置
    for i in range(10):
        if data.N[i]>0:
```

```
            choice=[data.l[i],data.w[i],data.h[i]] #六种摆放方式
            for j in choice:
                for k in choice:
                    for l in choice:
                        if k!=j and k!=l:
                            if j<L_space and k<W_space and l<H_space:
                                if constraint1(m,i)==True and constraint2(m,i,location)==
                                    True:
                                    solution.Cim[m].append(i)
                                    solution.nameim[m].append(data.name[i])
                                    solution.xim[m].append(j)
                                    solution.yim[m].append(k)
                                    solution.zim[m].append(l)
                                    solution.vim[m].append(data.v[i])
                                    solution.mim[m].append(data.m[i])
                                    data.N[i]-=1
                                    #S.remove(location)
                                    flag=1
                                    break
        if flag==1:
            location1=[j,0,location[2],1]
            location2=[0,k,location[2],2]
            location3=[0,0,location[2]+l,3]
            front_load(m,location1,space)
            right_load(m,j,location2,space)
            up_load(m,j,k,location3,space)
    for i in range(2000):
        load(i)
```